**Amendments to the Specification:**

Please replace paragraph [0052] in its entirety with the following paragraph. The following replacement paragraph is marked to show changes made.

**[0052]** Figure 3B is a flow diagram of an exemplary process for implementing a read operation with a mutating interceptor in accordance with an embodiment of the present invention. At step 330 a chunk of raw data is read. The raw data is read in bytes of a predetermined amount. At step 332, the bytes of raw data are placed in an internal buffer. The process is repeated (step 334) until the internal buffer is full with N bytes. Preferably, N is greater than the predetermined number of bytes of raw data being read. At step 336, the next hierarchical document input token is read from the internal buffer. If this token is not a value token (step 340) then it is a token associated with either a begin or an end container node, and is copied unchanged into the internal buffer at step 342. If the token is a value token (step 340), then it is determined if the token satisfies any issued queries at step 338. If the value token does satisfy any queries, then the callbacks corresponding to the satisfied queries are invoked at step 346. The current value is provided to an internal buffer at step 348, and N bytes are removed from the internal buffer and copied into the consume buffer at step 344. If no queries are satisfied by the value token (step 338), then the value token is copied back into the internal buffer at step 342. N bytes are removed from the internal buffer and copied into the consume buffer at step 344.

Please replace paragraph [0075] in its entirety with the following paragraph. The following replacement paragraph is marked to show changes made.

**[0075]** Figure 9 is an exemplary process for executing a write request. The write operation, similar to the read operation, is processed by the query canonicalization procedure at step 912. At step 914, this is determined to be safe or unsafe. Safe writes are resolved in the write cache level. Thus, if deemed safe at step 914, the write cache is ~~queried~~ probed at step ~~918~~ 916. If the query is already cached in the write cache (step 918), then the associated value is replaced by the one supplied by the current request at step 922. Otherwise (step 918)

a new entry is created in the write cache, associating the given query and value at step 920. Unsafe writes are more complex due to the following liabilities. Unsafe writes should be shielded from the (safe) queries already existing in the write cache because they can alias the same value node. Application of these aliased queries to the data can result in lost write requests. Unsafe writes should be shielded from future safe read requests because, again, they might alias the same value node. In this case, the unsafe write is invisible to the read request. Two new re-writers are created at step 924 and step 926. The first new re-writer created at step 924 separates between the existing write requests and the new unsafe request. Note that the original read cache is discarded because its fidelity to the resulting stream can no longer be guaranteed. The second new re-writer created at step 926 is returned to the user and is superimposed on top of the second one to separate future reads from the unsafe write. Note that the original prefetch set is inherited by the new top-level re-writer assuming that the user is still interested in finding out the values for the contained queries.